How to Design an Efficient Pseudospectral Code

John C. Bowman

University of Alberta

June 11, 2019

www.math.ualberta.ca/∼bowman/talks

Acknowledgements: Malcolm Roberts (Université de Strasbourg)

Outline

- Dealiased Pseudospectral Method
- 2D and 3D Skeleton ProtoDNS Codes on GitHub
- Key Ingredients for an Efficient Pseudospectral Solver:
  - Hermitian Symmetry (velocity and vorticity fields are real)
  - Implicit Dealiasing
  - Basdevant Reduction
  - Hybrid OpenMP/MPI Parallelization
  - Adaptive Time Stepping
  - Dynamic Moment Averaging
  - Conservative Integration
  - Implementation of White-Noise Forcing
- Conclusions

Pseudospectral Method

- Pseudospectral simulations are a widely used numerical tool for the study of fluid turbulence:

  – fast $N \log N$ scaling for N modes.

  – spectral accuracy: more accurate than finite-difference or finite-element methods.

- Ideal choice for studying homogenous turbulence with periodic boundary conditions.

- Generalizations such as Chebyshev collocation and penalty methods allow them to handle more complicated boundary conditions and geometries.

- However, in many cases pseudospectral methods do not parallelize well on massively parallel distributed architectures due to the communication costs of the parallel transpose.

Dealiasing

- Patterson and Orszag pioneered the pseudospectral method over 40 years ago.
- They emphasized that the convolution theorem necessitates *dealiasing* unwanted harmonics arising from the periodicity of the discrete Fourier transform.

DNS code

- We have released a highly optimized 2D pseudospectral code in C++: `https://github.com/dealias/dns`.
- It uses our `FFTW++` library to implicitly dealias the advective convolution, while exploiting Hermitian symmetry [Bowman & Roberts 2011], [Roberts & Bowman 2018].
- Advanced computer memory management, such as implicit padding, memory alignment, and dynamic moment averaging allow `DNS` to attain its extreme performance.
- The formulation proposed by Basdevant [1983] is used to reduce the number of FFTs required for 2D (3D) incompressible turbulence to 4 (8).
- We also include simplified 2D (146 lines) and 3D (287 lines) versions called `ProtoDNS` for educational purposes: `https://github.com/dealias/dns/tree/master/protodns`.

Hermitian Symmetry

- In Fourier space, the reality of the velocity and scalar vorticity fields leads to the Hermitian symmetries

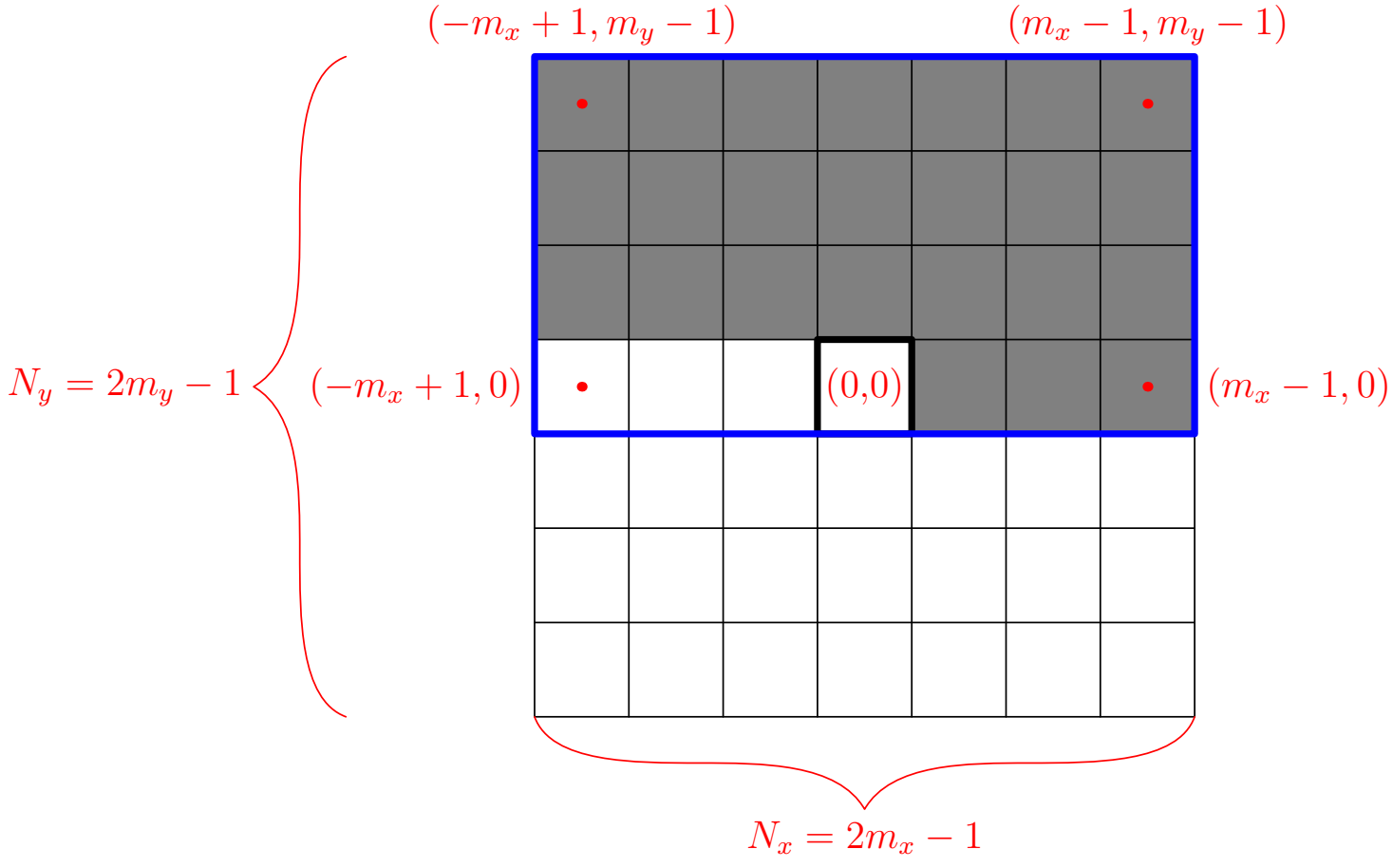
$$\boldsymbol{v}_{-\boldsymbol{k}} = \overline{\boldsymbol{v}_{\boldsymbol{k}}},$$

$$\omega_{-\boldsymbol{k}} = \overline{\omega_{\boldsymbol{k}}}.$$

- The DC mode at the Fourier origin is not evolved (no mean flow).

Hermitian Symmetry in 2D

- Only the shaded modes need to be evolved:



- Warning: since Hermitian convolution routines require input from the entire blue rectangle, the white modes in the blue box must agree with their shaded Hermitian conjugates!

Discrete Cyclic Convolution

- The FFT is an efficient tool for computing the *discrete cyclic convolution* of two vectors vectors F and G with period N:

$$\sum_{p=0}^{N-1} F_p G_{k-p}.$$

- But the pseudospectral method requires a *linear convolution!*

- The backward 1D *discrete Fourier transform* of a complex vector $\{F_k : k = 0, \ldots, N-1\}$ is defined as

$$f_j \doteq \sum_{k=0}^{N-1} \zeta_N^{jk} F_k, \qquad j = 0, \ldots, N-1,$$

  where $\zeta_N = e^{2\pi i/N}$ denotes the *Nth primitive root of unity*.

- The fast Fourier transform (FFT) method exploits the properties that $\zeta_N^r = \zeta_{N/r}$ and $\zeta_N^N = 1$.

Convolution Theorem

$$\begin{aligned}\sum_{j=0}^{N-1} f_j g_j \zeta_N^{-jk} &= \sum_{j=0}^{N-1} \zeta_N^{-jk} \left(\sum_{p=0}^{N-1} \zeta_N^{jp} F_p \right) \left(\sum_{q=0}^{N-1} \zeta_N^{jq} G_q \right) \\ &= \sum_{p=0}^{N-1} \sum_{q=0}^{N-1} F_p G_q \sum_{j=0}^{N-1} \zeta_N^{(-k+p+q)j} \\ &= N \sum_{s} \sum_{p=0}^{N-1} F_p G_{k-p+sN}.\end{aligned}$$

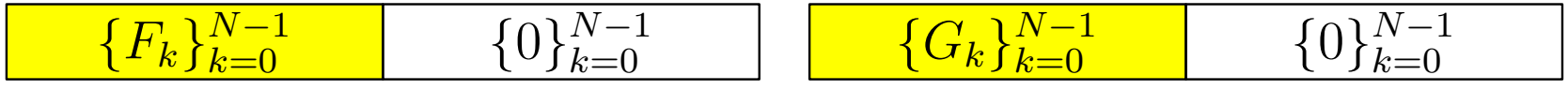
- The terms indexed by $s \neq 0$ are *aliases;* we need to remove them!
- If only the first m entries of the input vectors are nonzero, aliases can be avoided by *zero padding* input data vectors of length m to length $N \geq 2m - 1$.

Explicit Dealiasing

- *Explicit zero padding* prevents mode $m-1$ from beating with itself, wrapping around to contaminate mode $N = 0 \bmod N$:

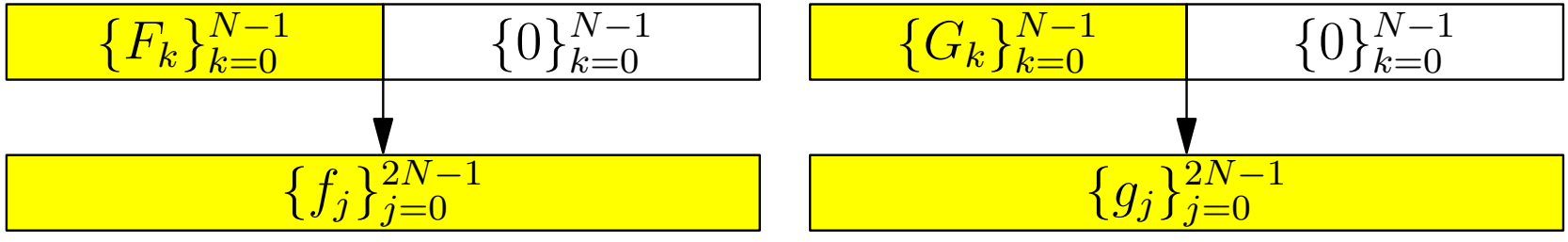
Explicit Dealiasing

- *Explicit zero padding* prevents mode $m-1$ from beating with itself, wrapping around to contaminate mode $N = 0 \bmod N$:



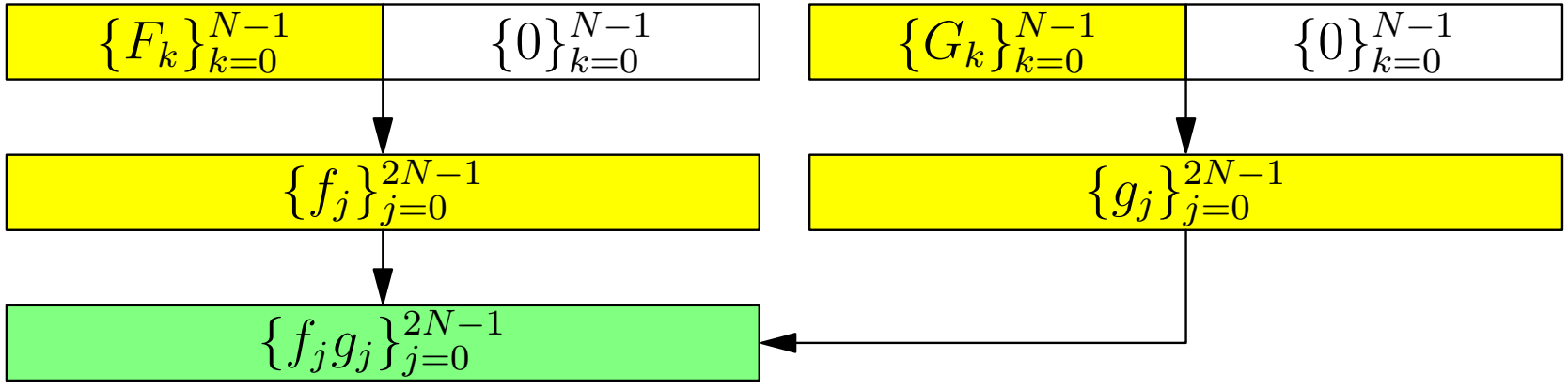
Explicit Dealiasing

- *Explicit zero padding* prevents mode $m-1$ from beating with itself, wrapping around to contaminate mode $N = 0 \bmod N$:



Explicit Dealiasing

- *Explicit zero padding* prevents mode $m-1$ from beating with itself, wrapping around to contaminate mode $N = 0 \bmod N$:



Implicit Dealiasing

- Let $N = 2m$. For $j = 0, \ldots, 2m-1$ we want to compute

$$f_j = \sum_{k=0}^{2m-1} \zeta_{2m}^{jk} F_k.$$

- If $F_k = 0$ for $k \geq m$, one can easily avoid looping over the unwanted zero Fourier modes by decimating in wavenumber:

$$f_{2\ell} = \sum_{k=0}^{m-1} \zeta_{2m}^{2\ell k} F_k = \sum_{k=0}^{m-1} \zeta_m^{\ell k} F_k,$$

$$f_{2\ell+1} = \sum_{k=0}^{m-1} \zeta_{2m}^{(2\ell+1)k} F_k = \sum_{k=0}^{m-1} \zeta_m^{\ell k} \zeta_{2m}^{k} F_k, \qquad \ell = 0, 1, \ldots m-1.$$

- This requires computing two subtransforms, each of size m, for an overall computational scaling of order $2m \log_2 m = N \log_2 m$.

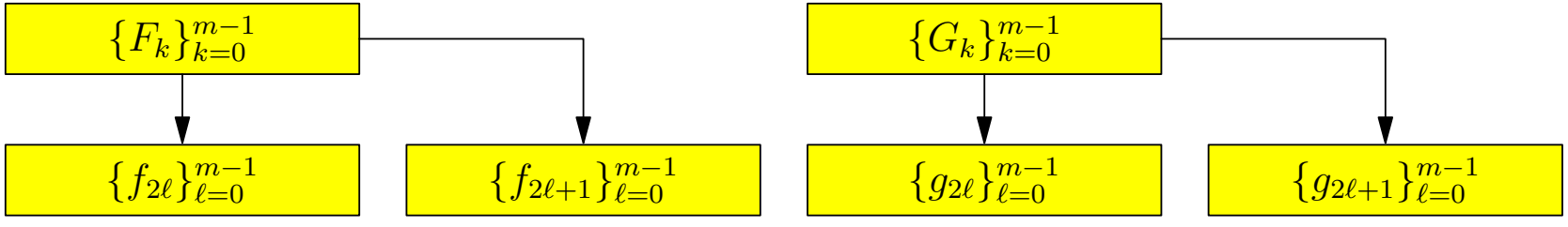
- Parallelized multidimensional implicit dealiasing routines have been implemented as a software layer `FFTW++` (v2.05) on top of the `FFTW` library under the Lesser GNU Public License:

  `http://fftwpp.sourceforge.net/`

$\{F_k\}_{k=0}^{m-1}$ $\{G_k\}_{k=0}^{m-1}$

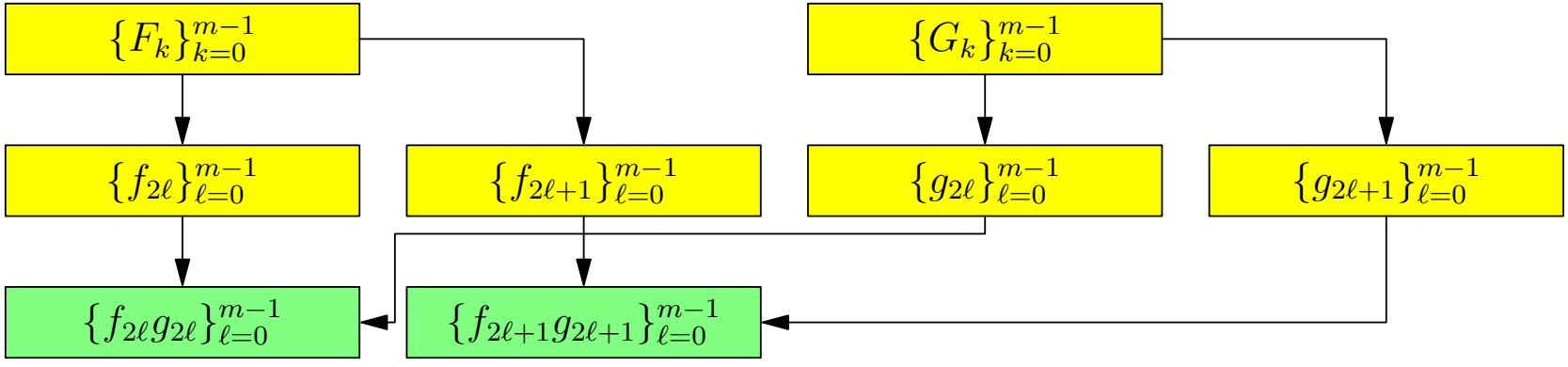
- Parallelized multidimensional implicit dealiasing routines have been implemented as a software layer `FFTW++` (v2.05) on top of the `FFTW` library under the Lesser GNU Public License:

  `http://fftwpp.sourceforge.net/`



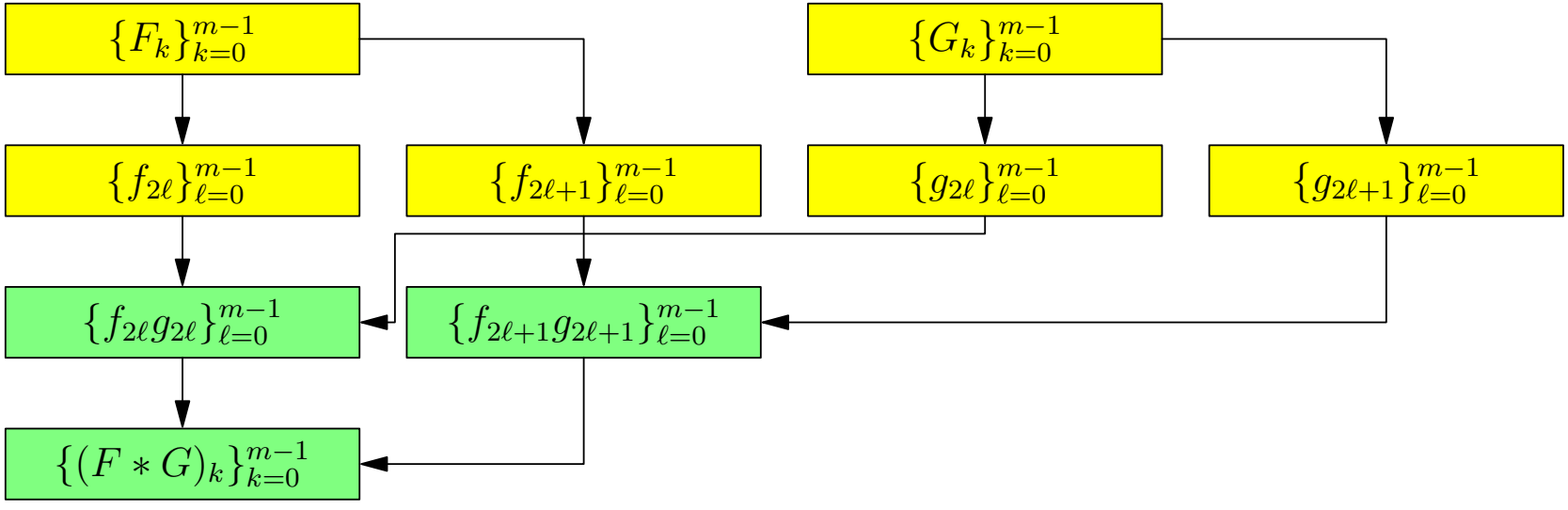
- Parallelized multidimensional implicit dealiasing routines have been implemented as a software layer **FFTW++** (v2.05) on top of the **FFTW** library under the Lesser GNU Public License:

  http://fftwpp.sourceforge.net/



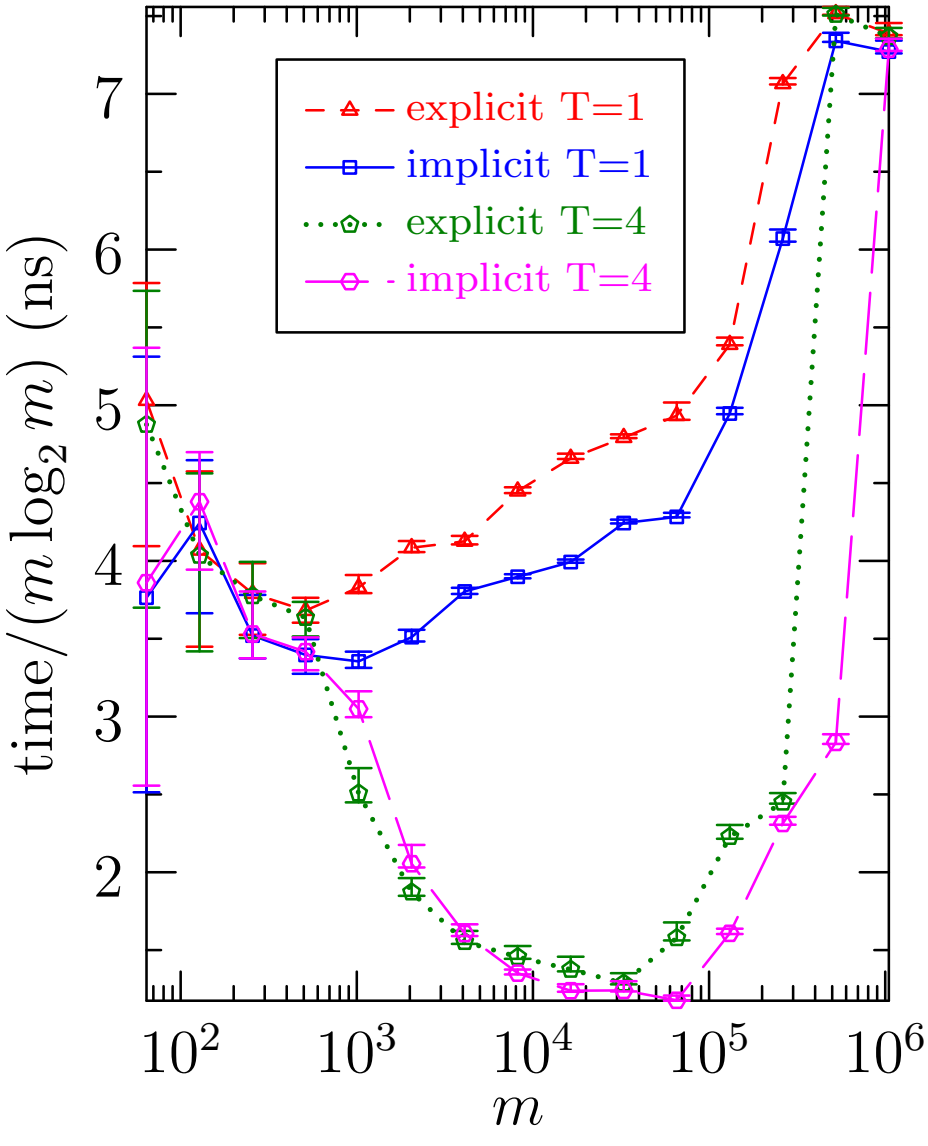
- Parallelized multidimensional implicit dealiasing routines have been implemented as a software layer **FFTW++** (v2.05) on top of the **FFTW** library under the Lesser GNU Public License:

  http://fftwpp.sourceforge.net/



Input: vector f, vector g
Output: vector f
$\mathsf{u} \leftarrow \mathtt{fft}^{-1}(\mathsf{f})$;
$\mathsf{v} \leftarrow \mathtt{fft}^{-1}(\mathsf{g})$;
$\mathsf{u} \leftarrow \mathsf{u} * \mathsf{v}$;
for $k = 0$ **to** $m - 1$ **do**
  $\mathsf{f}[k] \leftarrow \zeta_{2m}^{k} \mathsf{f}[k]$;
  $\mathsf{g}[k] \leftarrow \zeta_{2m}^{k} \mathsf{g}[k]$;
end
$\mathsf{v} \leftarrow \mathtt{fft}^{-1}(\mathsf{f})$;
$\mathsf{f} \leftarrow \mathtt{fft}^{-1}(\mathsf{g})$;
$\mathsf{v} \leftarrow \mathsf{v} * \mathsf{f}$;
$\mathsf{f} \leftarrow \mathtt{fft}(\mathsf{u})$;
$\mathsf{u} \leftarrow \mathtt{fft}(\mathsf{v})$;
for $k = 0$ **to** $m - 1$ **do**
  $\mathsf{f}[k] \leftarrow \mathsf{f}[k] + \zeta_{2m}^{-k} \mathsf{u}[k]$;
end
return f/(2m);

Implicit Padding in 1D on T threads



Convolutions in Higher Dimensions

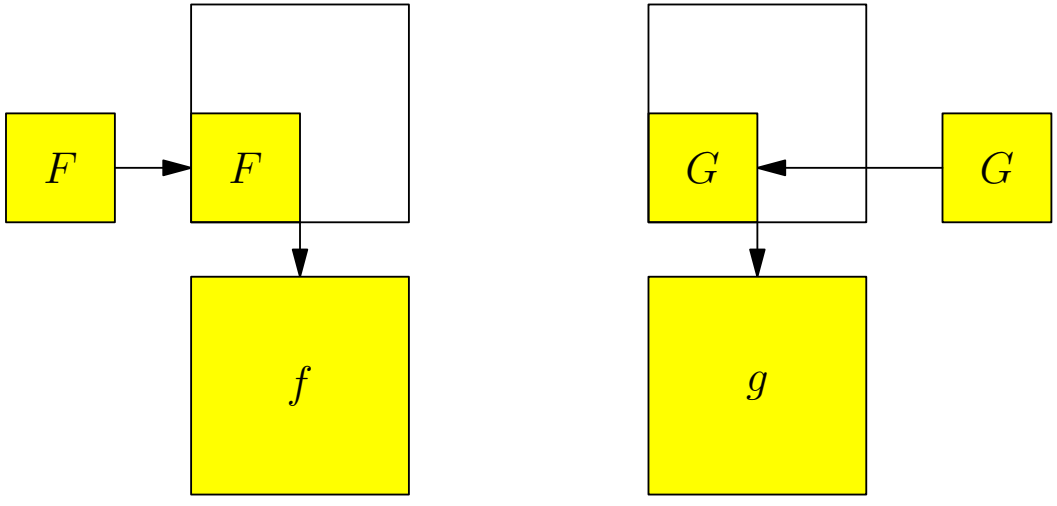
- An explicitly padded convolution in 2 dimensions requires 12 padded FFTs and 4 times the memory of a cyclic convolution.

Convolutions in Higher Dimensions

- An explicitly padded convolution in 2 dimensions requires 12 padded FFTs and 4 times the memory of a cyclic convolution.

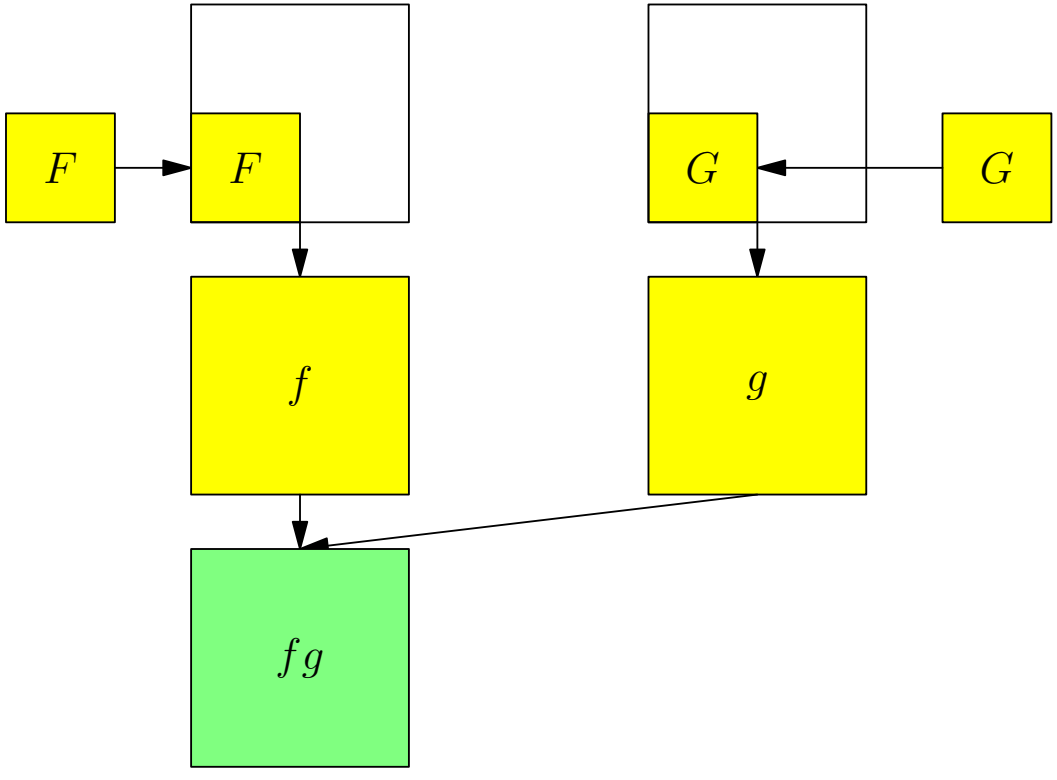
Convolutions in Higher Dimensions

- An explicitly padded convolution in 2 dimensions requires 12 padded FFTs and 4 times the memory of a cyclic convolution.



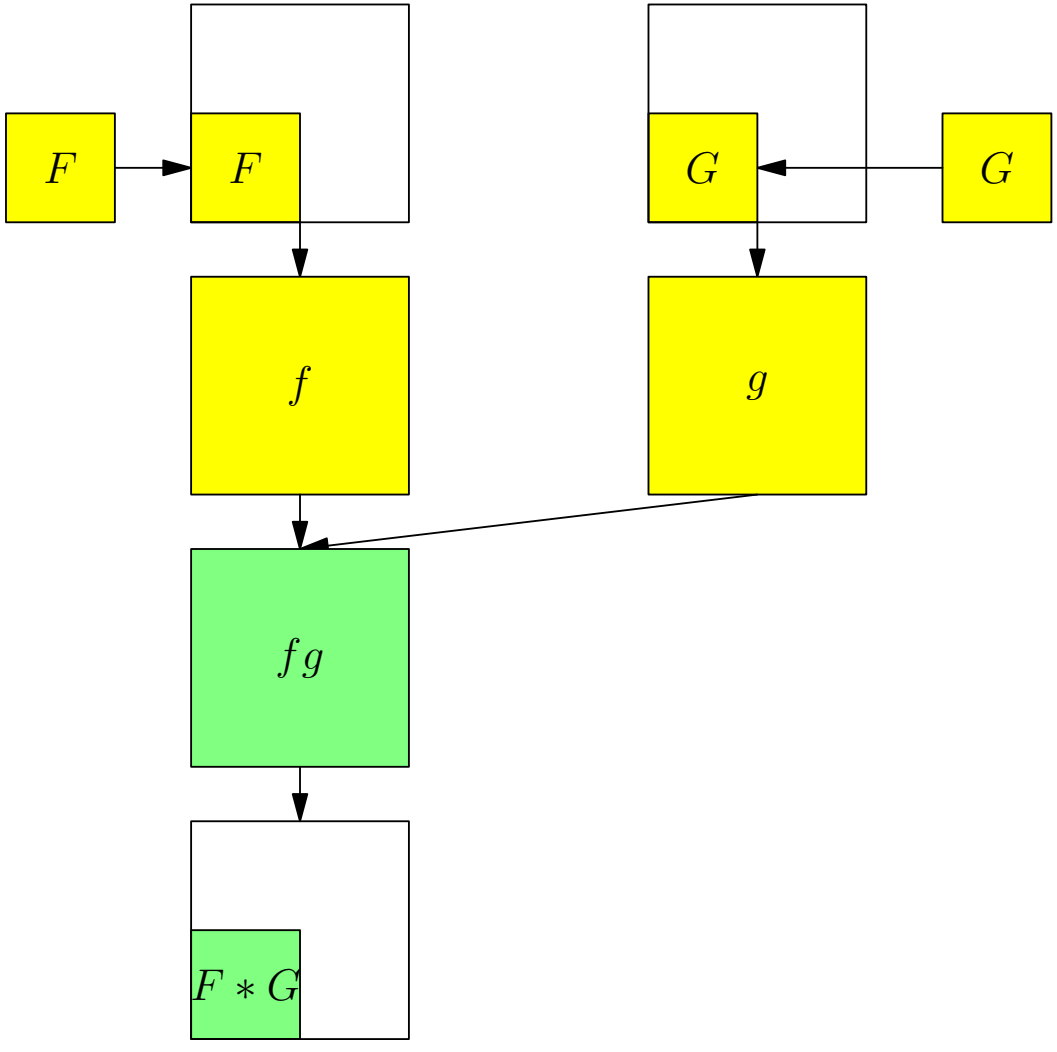
Convolutions in Higher Dimensions

- An explicitly padded convolution in 2 dimensions requires 12 padded FFTs and 4 times the memory of a cyclic convolution.



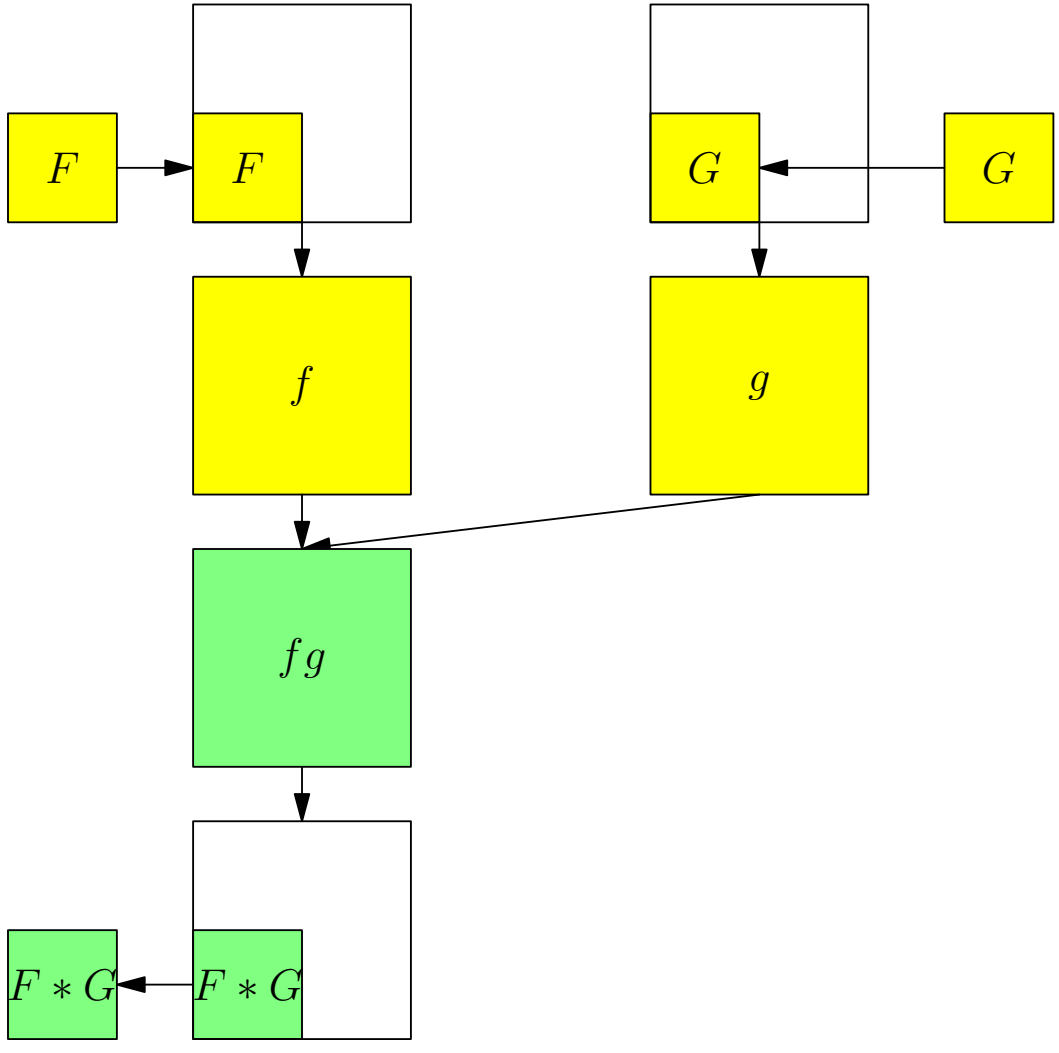
Convolutions in Higher Dimensions

- An explicitly padded convolution in 2 dimensions requires 12 padded FFTs and 4 times the memory of a cyclic convolution.



Convolutions in Higher Dimensions

- An explicitly padded convolution in 2 dimensions requires 12 padded FFTs and 4 times the memory of a cyclic convolution.



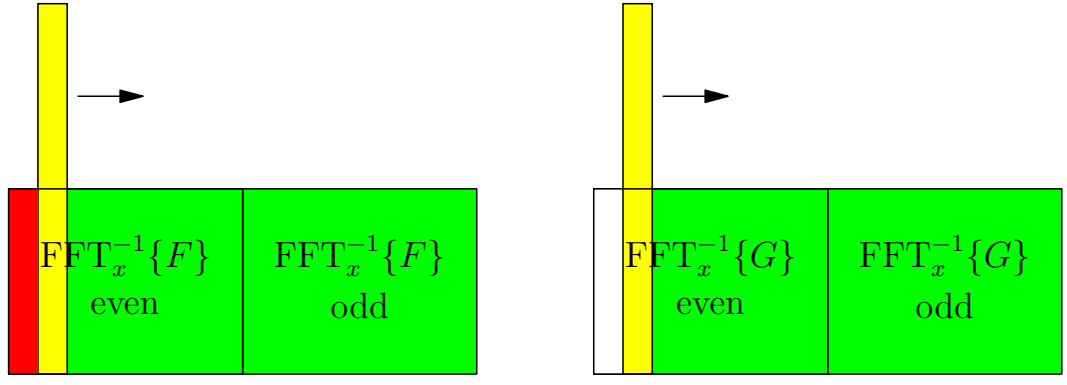
Recursive Convolution

- Naive way to compute a multiple-dimensional convolution:

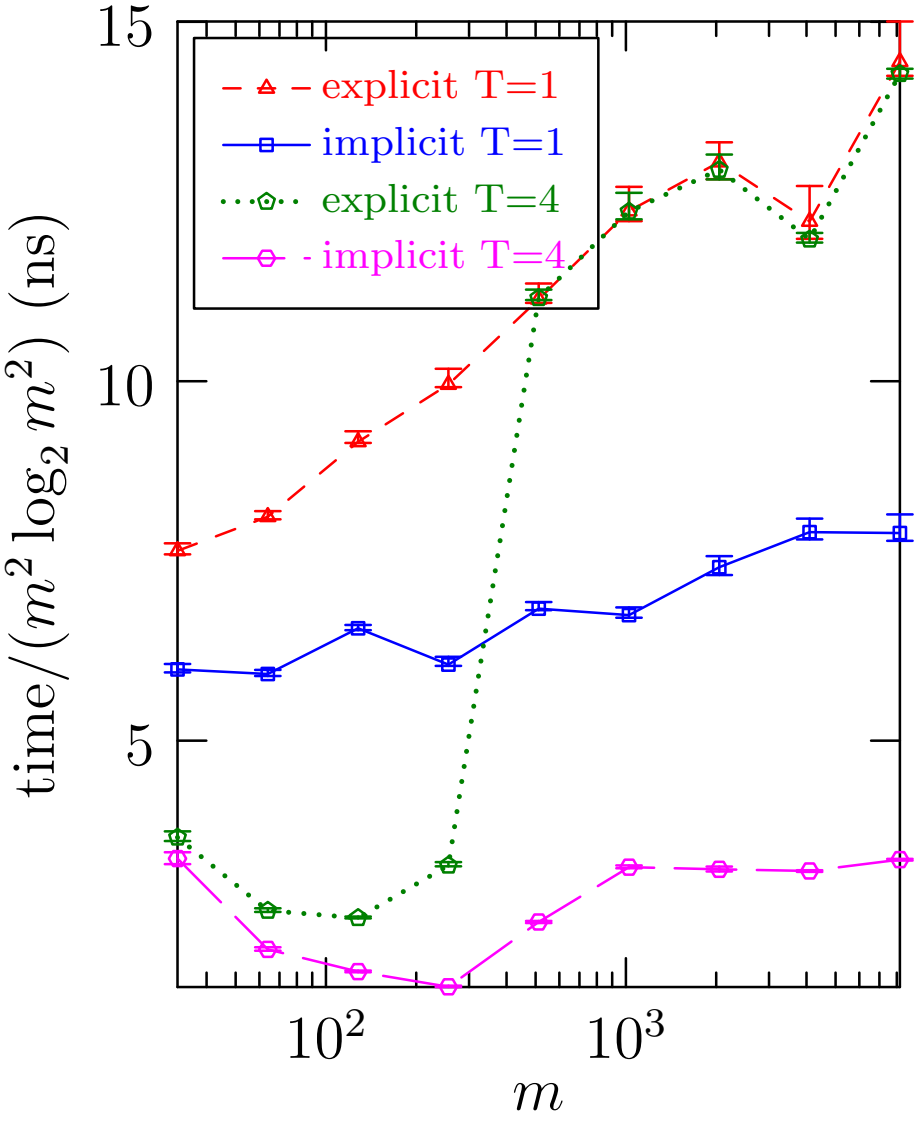
- The technique of *recursive convolution* allows one to avoid computing and storing the entire Fourier image of the data:

Implicit Padding in 2D

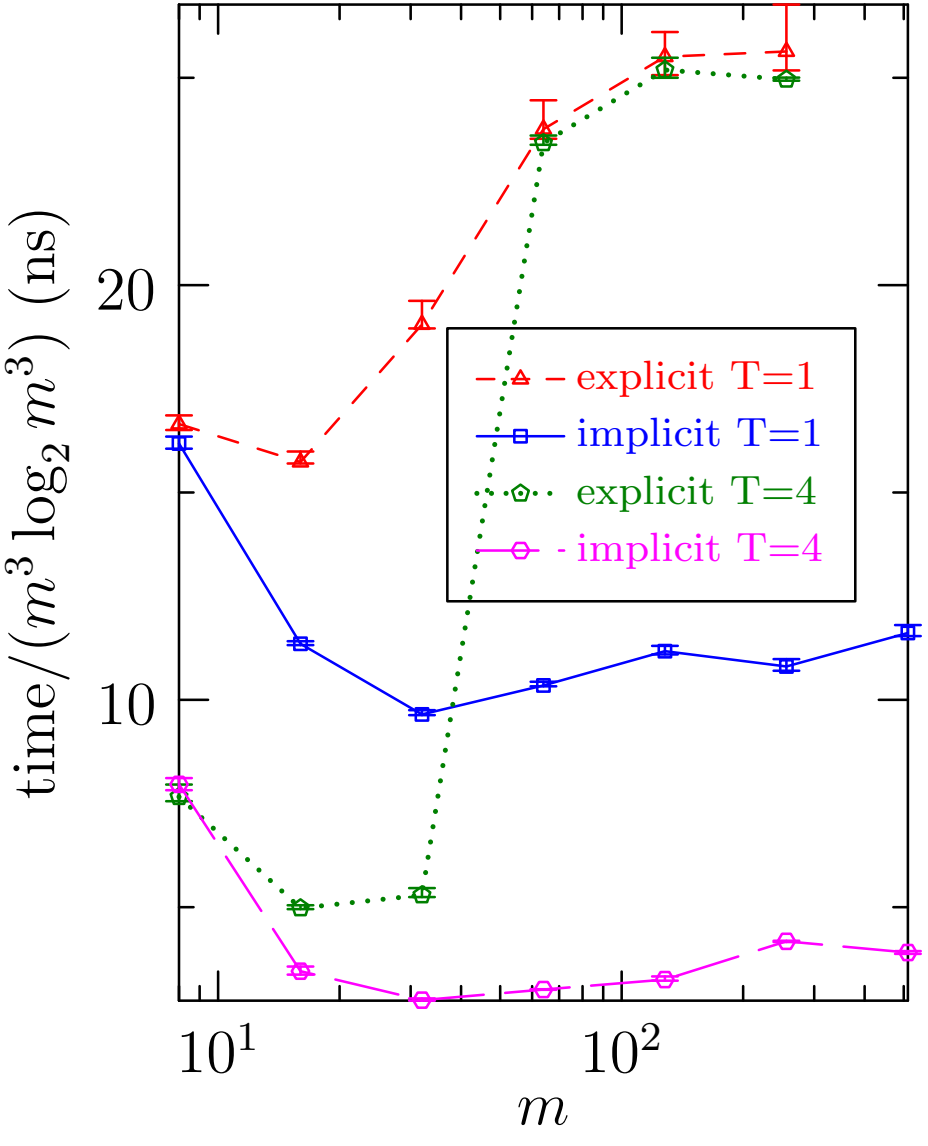
- Extra work memory need not be contiguous with the data.



Implicit Padding in 2D on T threads



Implicit Padding in 3D on T threads



Centered (Pseudospectral) Convolutions

- For a *centered convolution,* the Fourier origin ($k = 0$) is centered in the domain:

$$\sum_{p=k-m+1}^{m-1} f_p g_{k-p}$$

- Need to pad to $N \geq 3m - 2$ to remove aliases.
- The ratio $(2m-1)/(3m-2)$ of the number of physical to total modes is asymptotic to $2/3$ for large m.
- A *Hermitian convolution* arises since the input vectors are real:

$$f_{-\boldsymbol{k}} = \overline{f_{\boldsymbol{k}}}.$$

Hermitian Convolution

- The backwards implicitly padded centered Hermitian transform appears as

$$u_{3\ell+r} = \sum_{k=0}^{m-1} \zeta_m^{\ell k} w_{k,r},$$

where

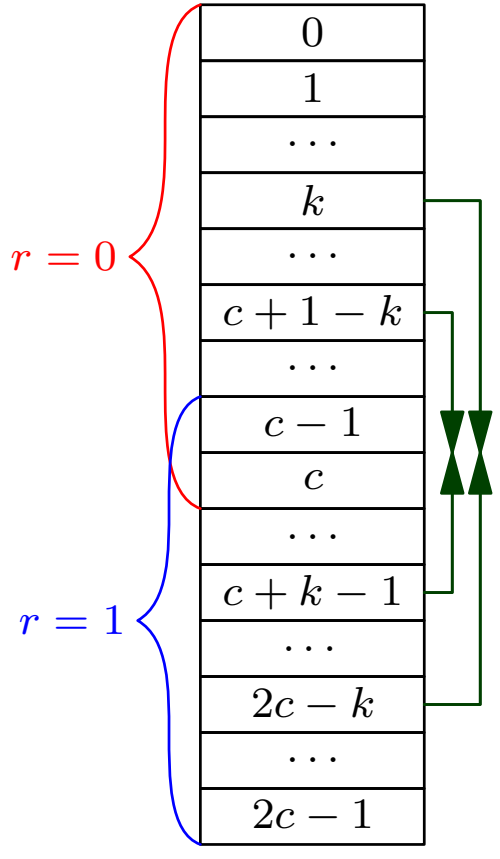
$$w_{k,r} \doteq \begin{cases} U_0 + \operatorname{Re} \zeta_3^{-r} U_{-m} & \text{if } k = 0, \\ \zeta_{3m}^{rk} \left(U_k + \zeta_3^{-r} \overline{U_{m-k}} \right) & \text{if } 1 \le k \le m-1. \end{cases}$$

- We exploit the Hermitian symmetry $w_{k,r} = \overline{w_{m-k,r}}$ to reduce the problem to three complex-to-real Fourier transforms of the first $c+1$ components of $w_{k,r}$ (one for each $r = -1, 0, 1$), where $c \doteq \lfloor m/2 \rfloor$ zeros.

- To facilitate an in-place implementation, in our original paper [SIAM J. Sci. Comput. 33, 386 (2011)], we stored the transformed values for $r = 1$ in reverse order in the upper half of the input vector.
- However, loop dependencies in the resulting algorithm prevented the top level of the 1D transforms from being multithreaded.

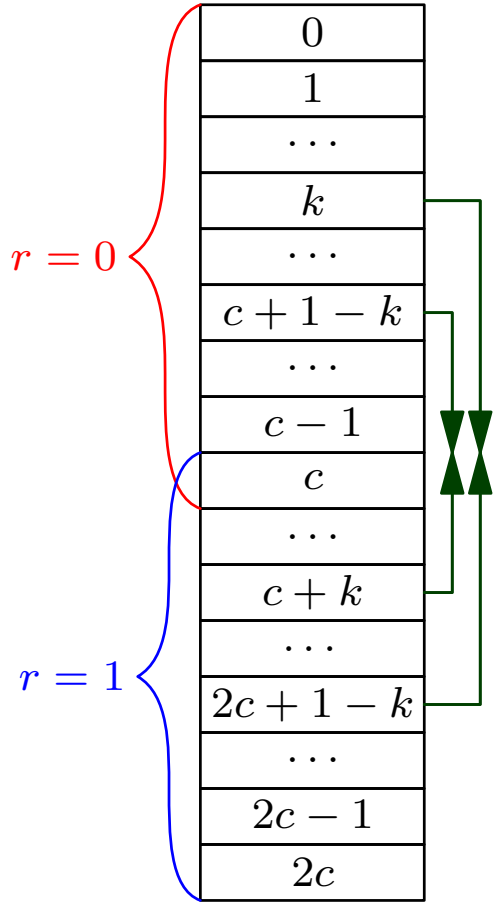
Multithreaded Hermitian Convolution

- Unrolling the loop to process four inputs and outputs simultaneously allows loop independence to be achieved, significantly improving performance in both the serial and parallel contexts.



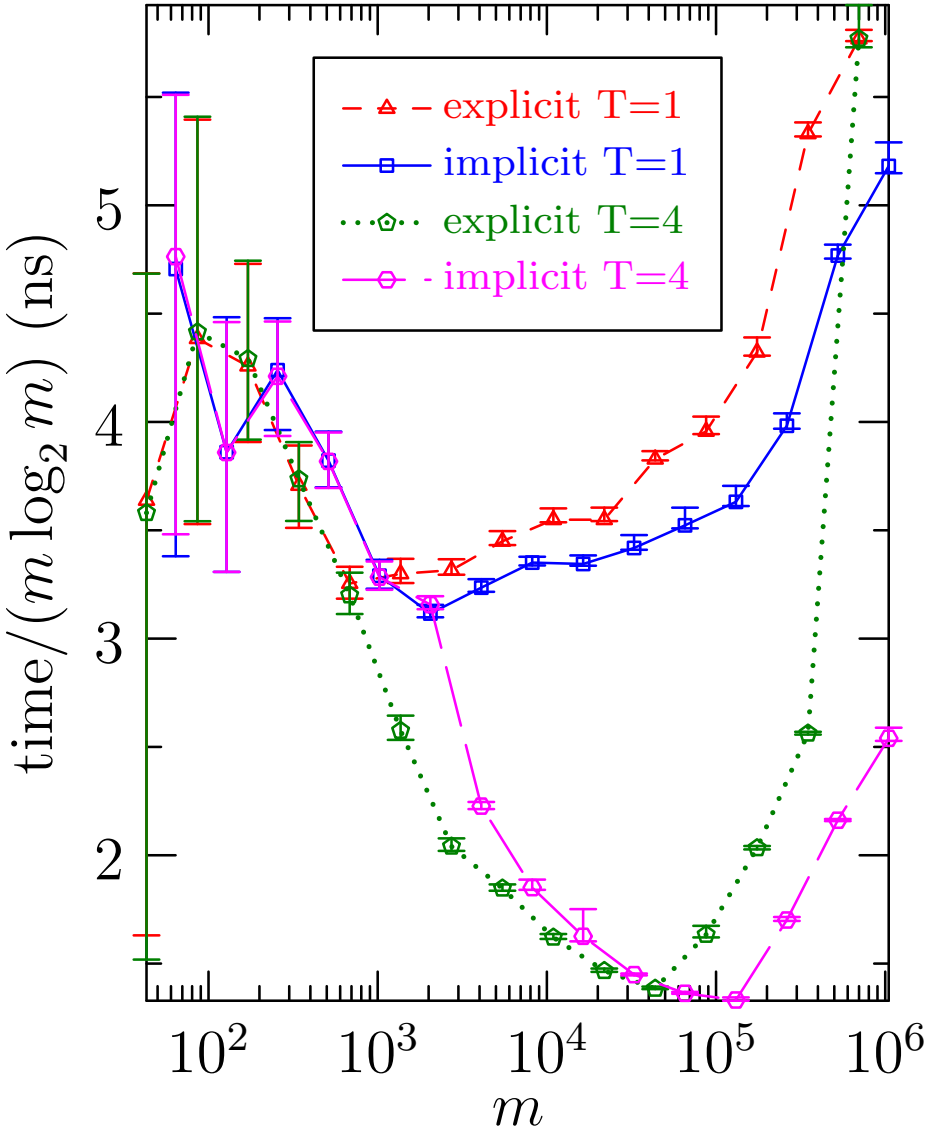
Multithreaded Hermitian Convolution

- Unrolling the loop to process four inputs and outputs simultaneously allows loop independence to be achieved, significantly improving performance in both the serial and parallel contexts.



- As a result, even in 1D, implicit dealiasing of pseudospectral convolutions is now significantly faster than explicit zero padding.

1D Implicit Hermitian Convolution



3D Basdevant Reduction: 8 FFTs

- Using incompressibility, the 3D momentum equation can be written in terms of the symmetric tensor $D_{ij} = u_i u_j$:

$$\frac{\partial u_i}{\partial t} + \frac{\partial D_{ij}}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \nu \frac{\partial^2 u_i}{\partial x_j^2} + F_i.$$

- Naive implementation: 3 backward FFTs to compute the velocity components from their spectral representations, 6 forward FFTs of the independent components of D_{ij}.

- Basdevant [1983]: avoid one FFT by subtracting the divergence of the symmetric matrix $S_{ij} = \delta_{ij} \operatorname{tr} D/3$ from both sides:

$$\frac{\partial u_i}{\partial t} + \frac{\partial (D_{ij} - S_{ij})}{\partial x_j} = -\frac{\partial (p\delta_{ij} + S_{ij})}{\partial x_j} + \nu \frac{\partial^2 u_i}{\partial x_j^2} + F_i.$$

- To compute the velocity components u_i, 3 backward FFTs are required. Since the symmetric matrix $D_{ij} - S_{ij}$ is traceless, it has just 5 independent components.

- Hence, a total of only 8 FFTs are required per integration stage.
- The effective pressure $p\delta_{ij} + S_{ij}$ is solved as usual from the inverse Laplacian of the force minus the nonlinearity.

2D Basdevant Reduction: 4 FFTs

- The vorticity $\boldsymbol{w} = \boldsymbol{\nabla} \times \boldsymbol{u}$ evolves according to

$$\frac{\partial \boldsymbol{w}}{\partial t} + (\boldsymbol{u} \cdot \boldsymbol{\nabla})\boldsymbol{w} = (\boldsymbol{w} \cdot \boldsymbol{\nabla})\boldsymbol{u} + \nu \nabla^2 \boldsymbol{w} + \boldsymbol{\nabla} \times \boldsymbol{F},$$

  where in 2D the vortex stretching term $(\boldsymbol{w} \cdot \boldsymbol{\nabla})\boldsymbol{u}$ vanishes and $\boldsymbol{w}$ is normal to the plane of motion.

- For C^2 velocity fields, the curl of the nonlinearity can be written in terms of $\widetilde{D}_{ij} \doteq D_{ij} - S_{ij}$:

$$\frac{\partial}{\partial x_1}\frac{\partial}{\partial x_j}\widetilde{D}_{2j} - \frac{\partial}{\partial x_2}\frac{\partial}{\partial x_j}\widetilde{D}_{1j} = \left(\frac{\partial^2}{\partial x_1^2} - \frac{\partial^2}{\partial x_2^2}\right) D_{12} + \frac{\partial}{\partial x_1}\frac{\partial}{\partial x_2}(D_{22} - D_{11}),$$

  on recalling that S is diagonal and $S_{11} = S_{22}$.

- The scalar vorticity ω thus evolves as

$$\frac{\partial \omega}{\partial t} + \left(\frac{\partial^2}{\partial x_1^2} - \frac{\partial^2}{\partial x_2^2}\right)(u_1 u_2) + \frac{\partial^2}{\partial x_1 \partial x_2}\left(u_2^2 - u_1^2\right) = \nu \nabla^2 \omega + \frac{\partial F_2}{\partial x_1} - \frac{\partial F_1}{\partial x_2}.$$

- To compute u_1 and u_2 in physical space, we need 2 backward FFTs.
- The quantities $u_1 u_2$ and $u_2^2 - u_1^2$ can then be calculated and then transformed to Fourier space with 2 additional forward FFTs.
- The advective term in 2D can thus be calculated with just 4 FFTs.

3D Incompressible MHD: 14 FFTs

$$\frac{\partial u_i}{\partial t} + \frac{\partial (D_{ij} - S_{ij})}{\partial x_j} = -\frac{\partial (p\delta_{ij} + S_{ij})}{\partial x_j} + \nu \frac{\partial^2 u_i}{\partial x_j^2},$$

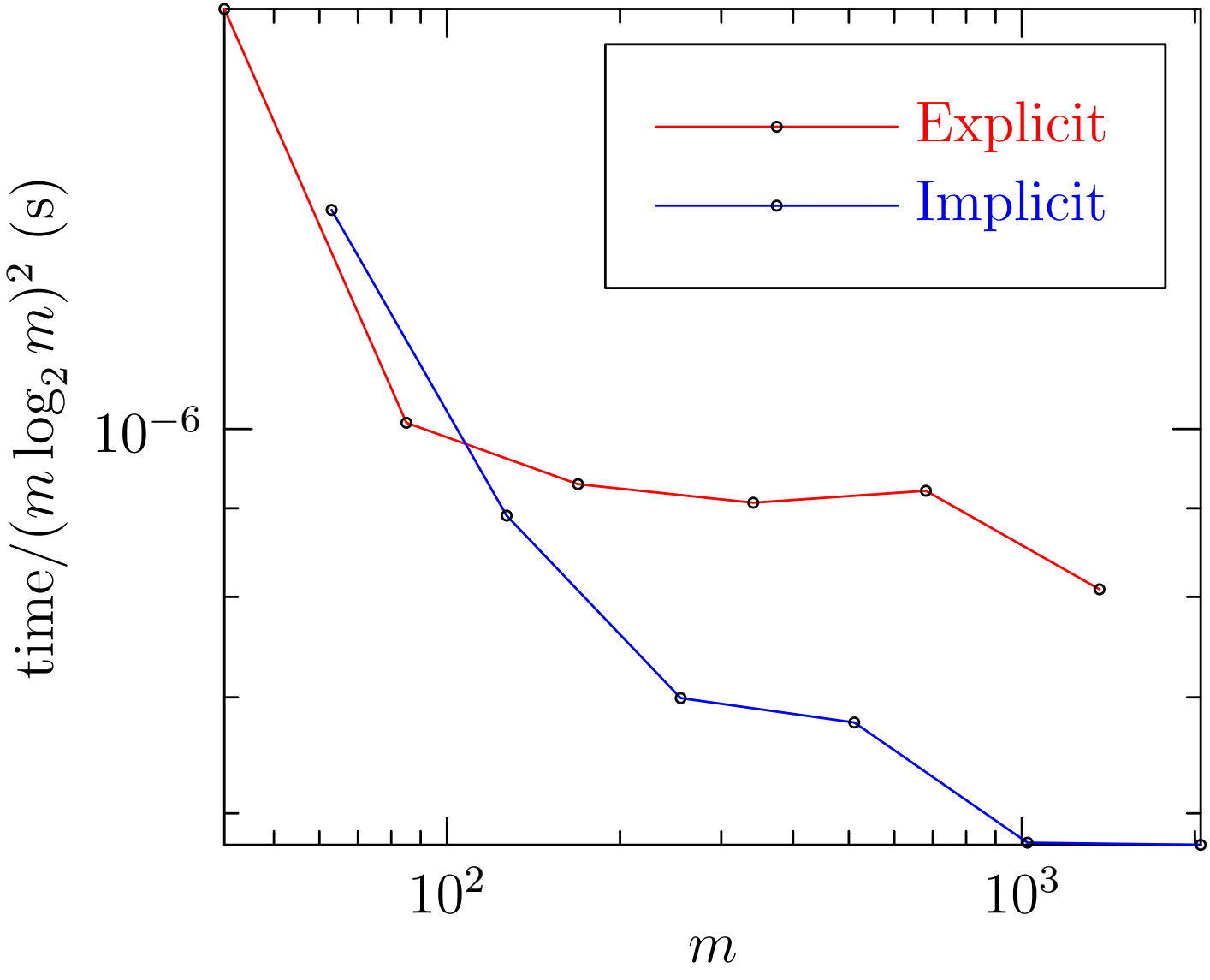
$$\frac{\partial B_i}{\partial t} + \frac{\partial G_{ij}}{\partial x_j} = \eta \frac{\partial^2 B_i}{\partial x_j^2},$$

where $D_{ij} = u_i u_j - B_i B_j$, $S_{ij} = \delta_{ij} \operatorname{tr} D/3$, and

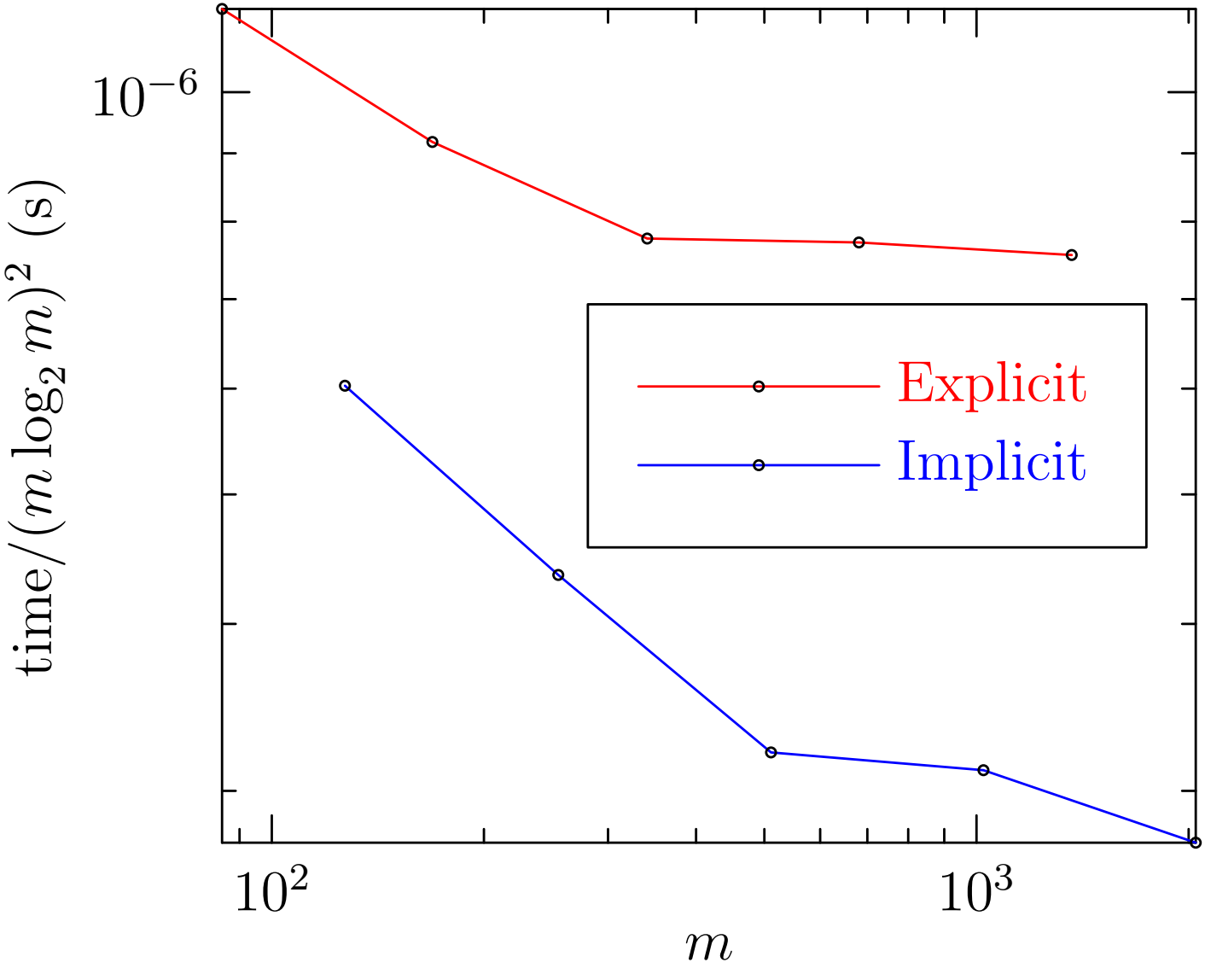
$$G_{ij} = B_i u_j - u_i B_j.$$

- The traceless symmetric matrix $D_{ij} - S_{ij}$ has 5 independent components.
- The antisymmetric matrix G_{ij} has only 3.
- An additional 6 FFT calls are required to compute the components of $\boldsymbol{u}$ and $\boldsymbol{B}$ in x space.
- The MHD nonlinearity can thus be computed with 14 FFT calls.

2D Navier–Stokes Pseudospectral [1 thread]



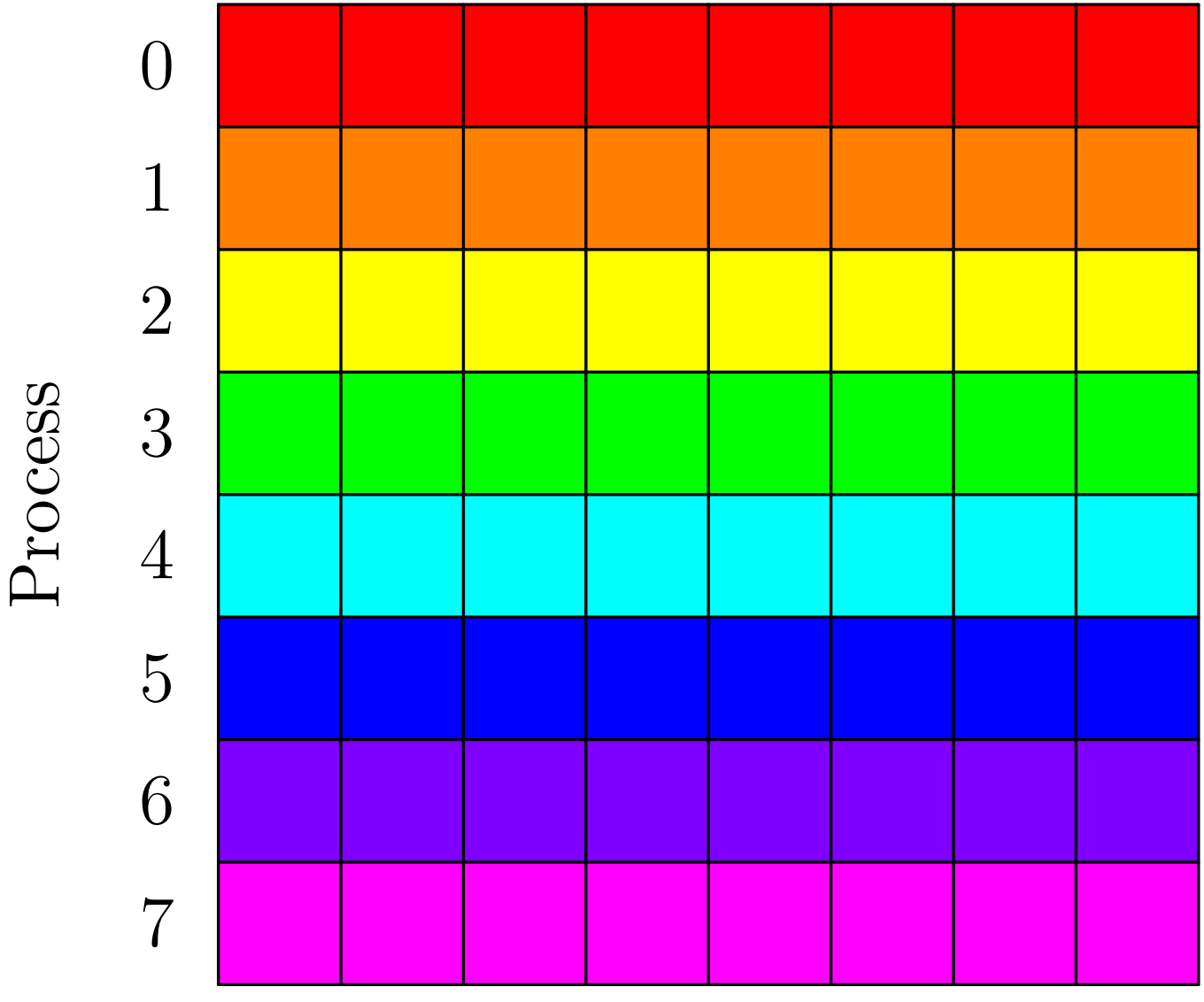
2D Navier–Stokes Pseudospectral [4 threads]



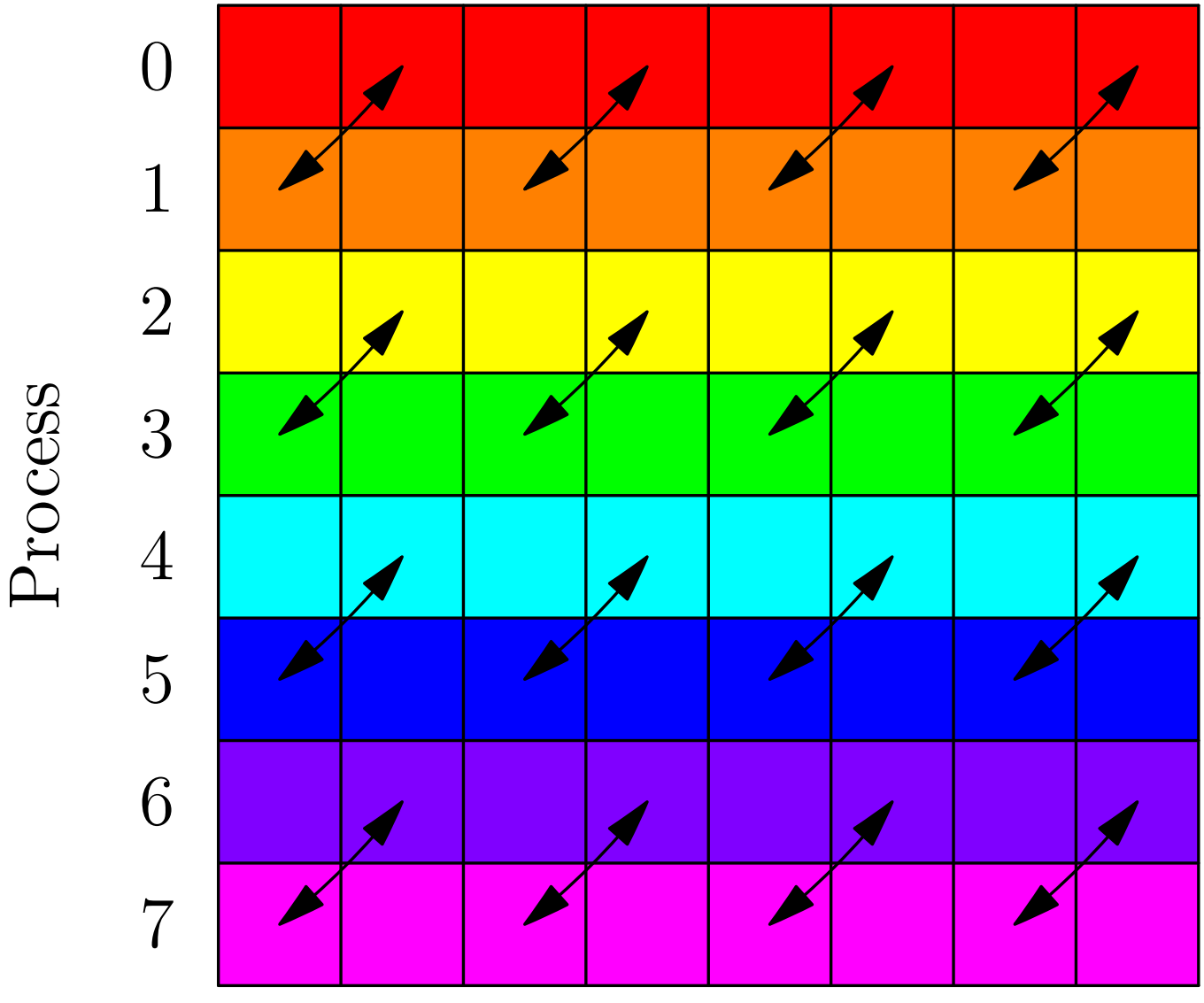
Distributed-Memory Parallelization

- The pseudospectral method uses a matrix transpose to localize the computation of the multi-dimensional FFTs onto individual processors.
- Parallel generalized slab/pencil decompositions have recently been developed for distributed-memory architectures.
- We have compared several distributed matrix transpose algorithms, both blocking and nonblocking, under pure MPI and hybrid MPI/OpenMP architectures.
- Local transposition is not required within a single MPI node.
- We have developed an adaptive algorithm, dynamically tuned to choose the optimal block size.

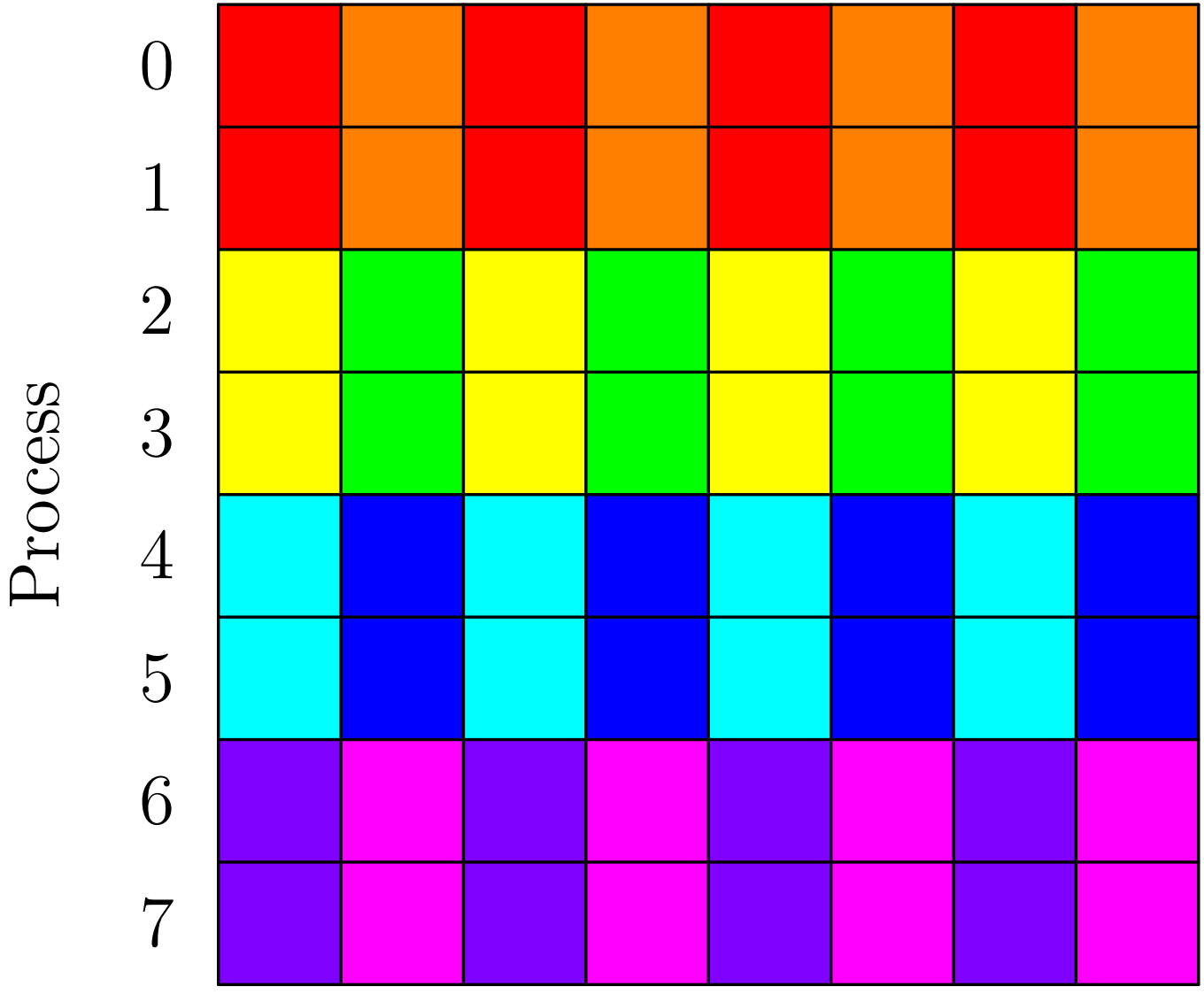
8×8 Block Transpose over 8 processors



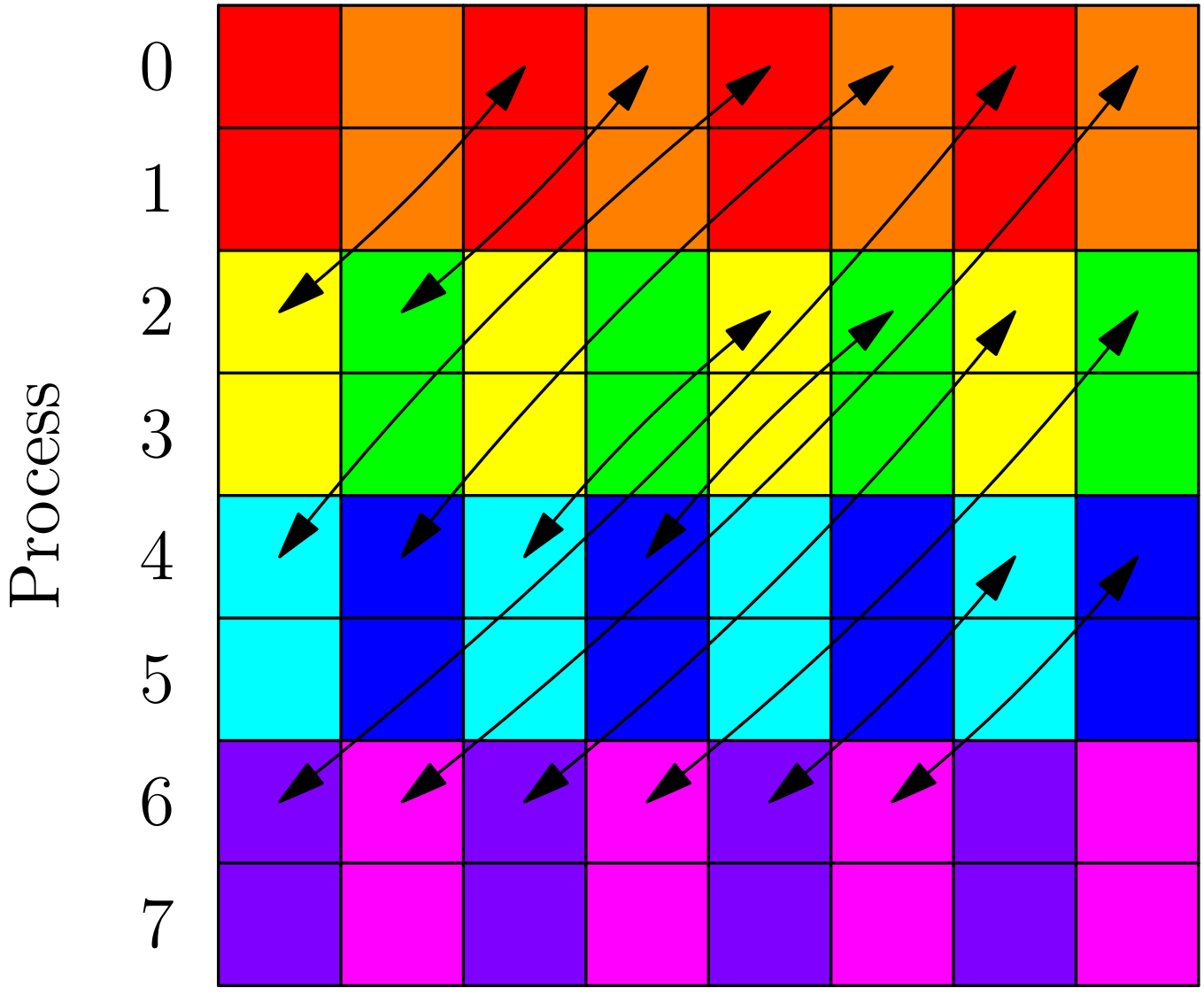
8×8 Block Transpose over 8 processors



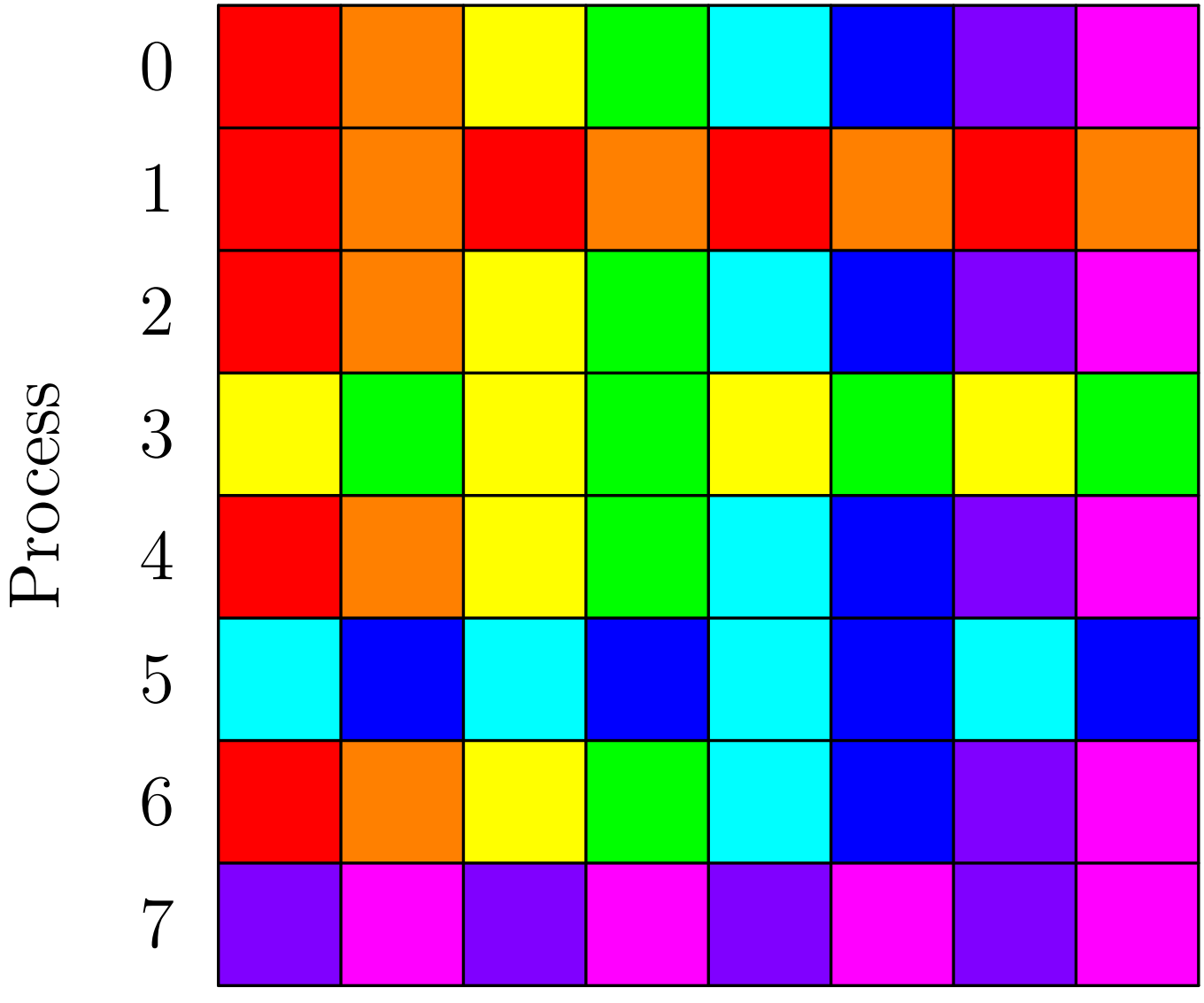
8×8 Block Transpose over 8 processors



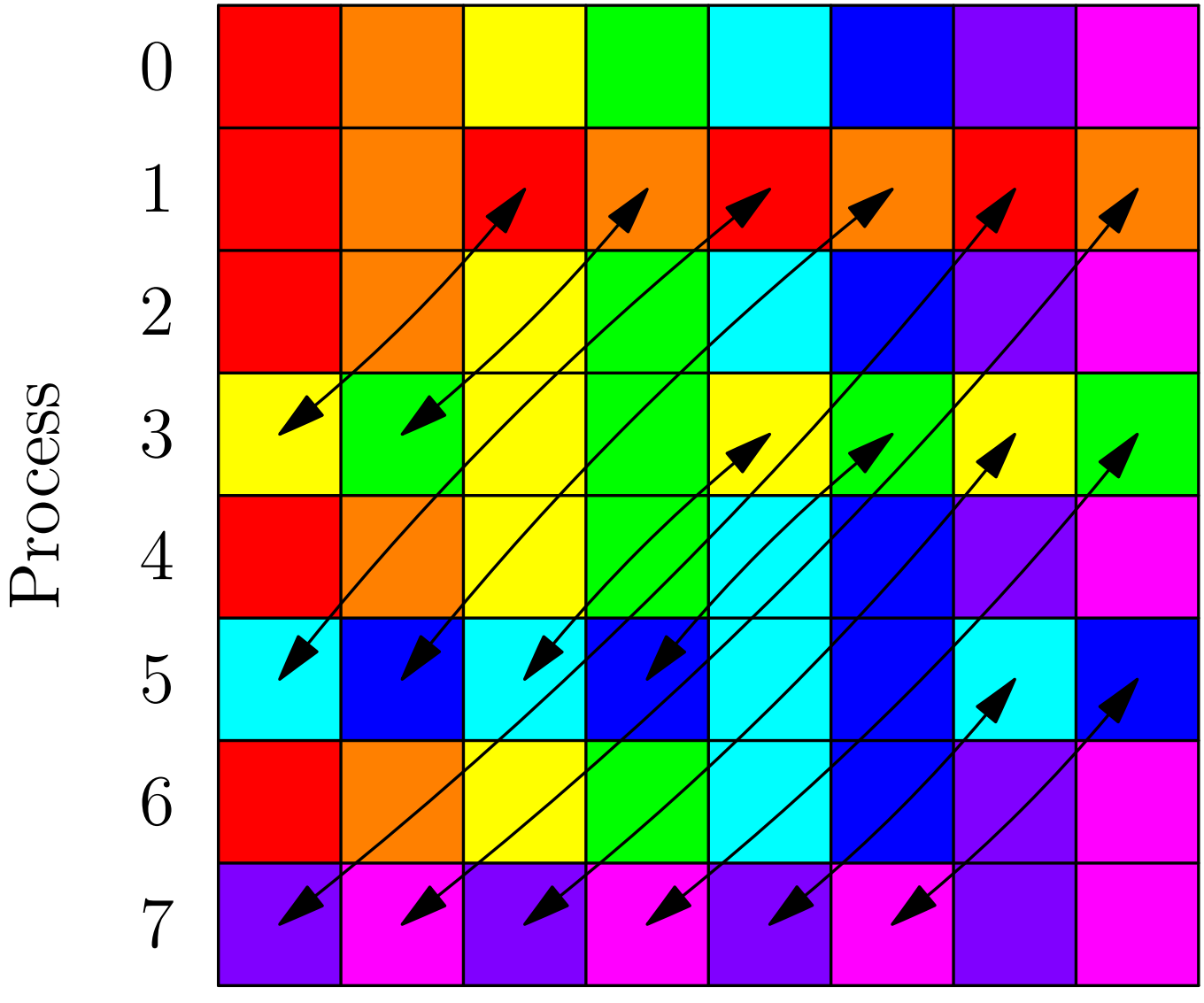
8×8 Block Transpose over 8 processors



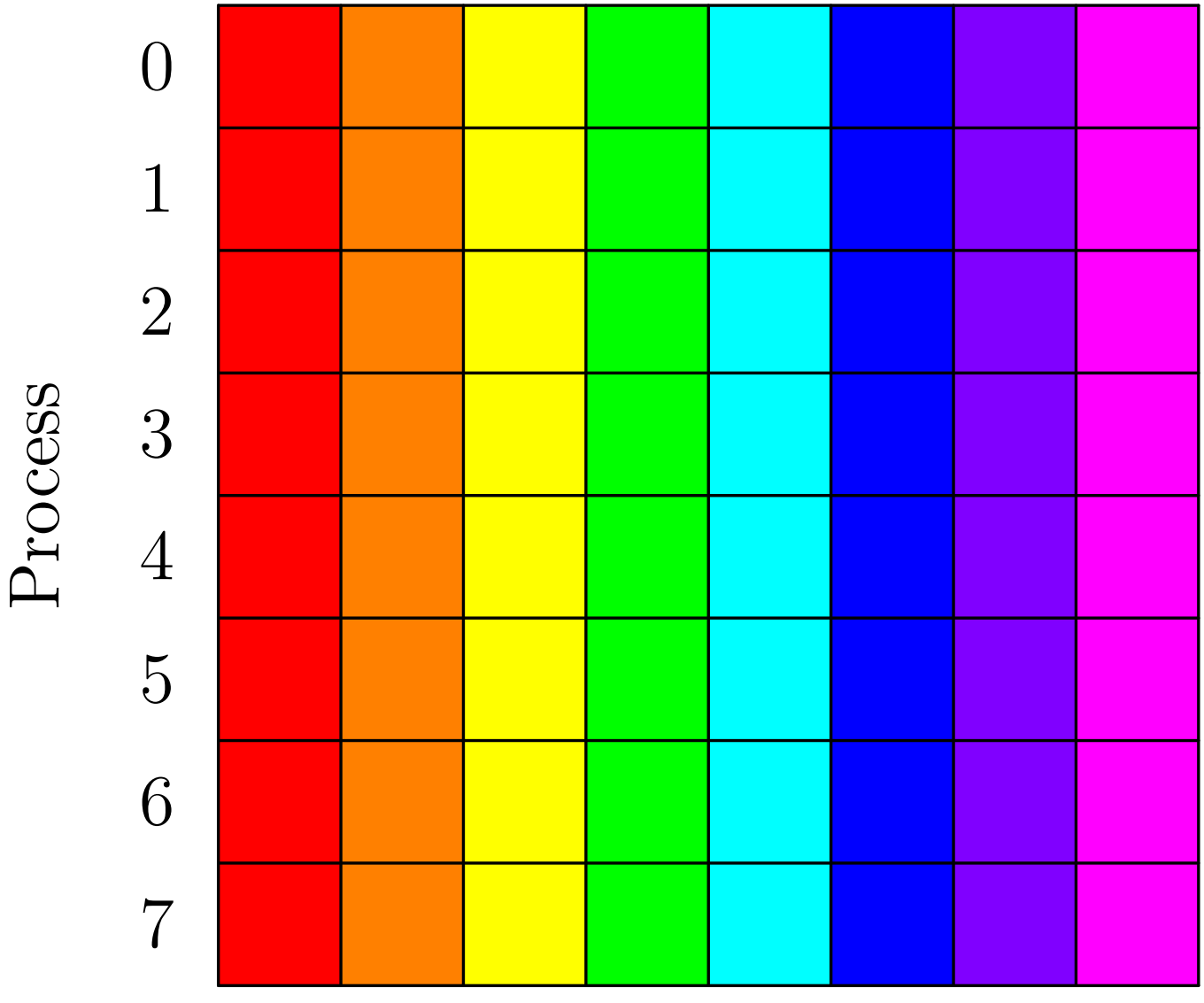
8×8 Block Transpose over 8 processors



8×8 Block Transpose over 8 processors



8×8 Block Transpose over 8 processors



Advantages of Hybrid MPI/OpenMP

- Use hybrid OpenMPI/MPI with the optimal number of threads:

  – yields larger communication block size;

  – local transposition is not required within a single MPI node;

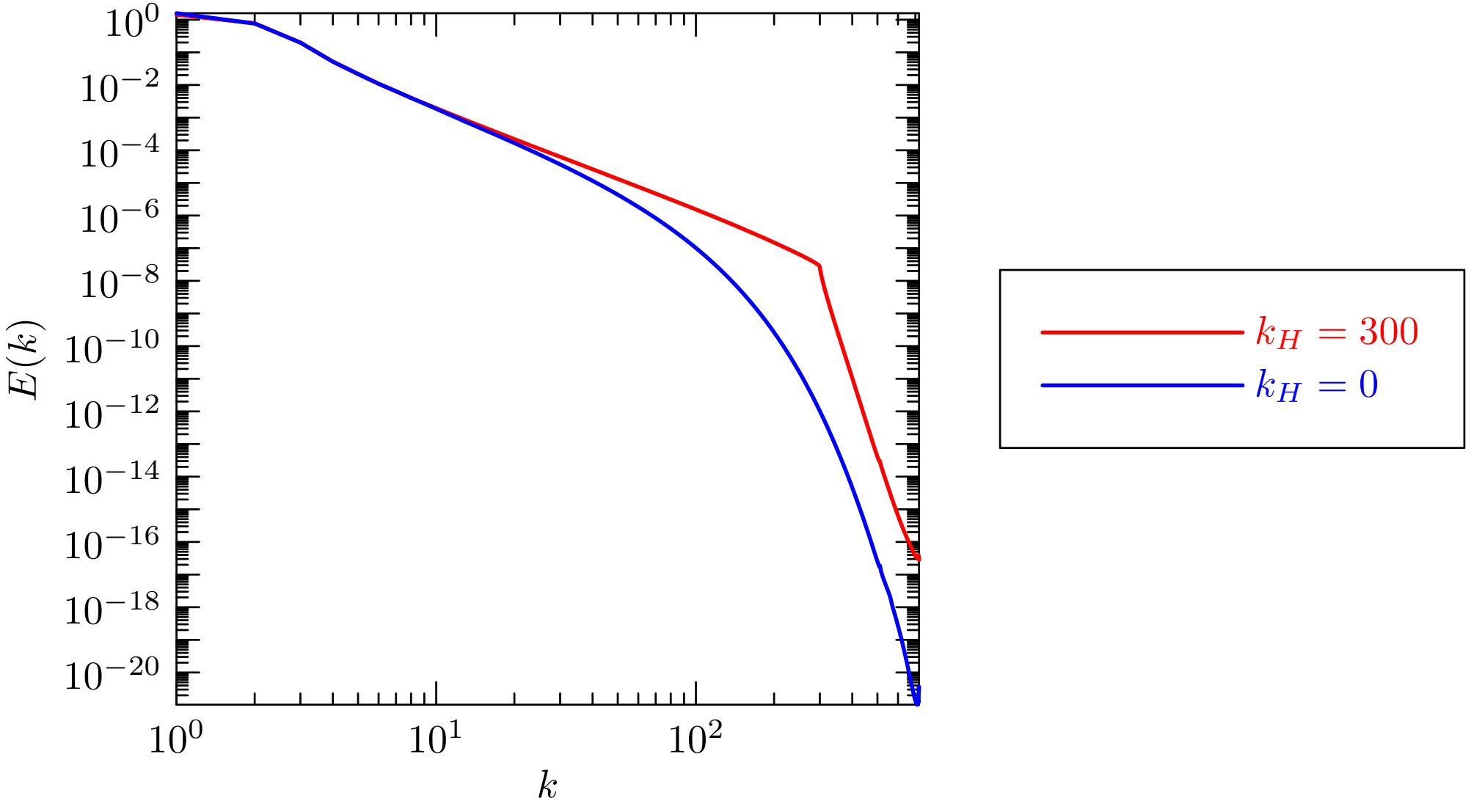
  – allows smaller problems to be distributed over a large number of processors;

  – for 3D FFTs, allows for more slab-like than pencil-like models, reducing the size of or even eliminating the need for a second transpose.

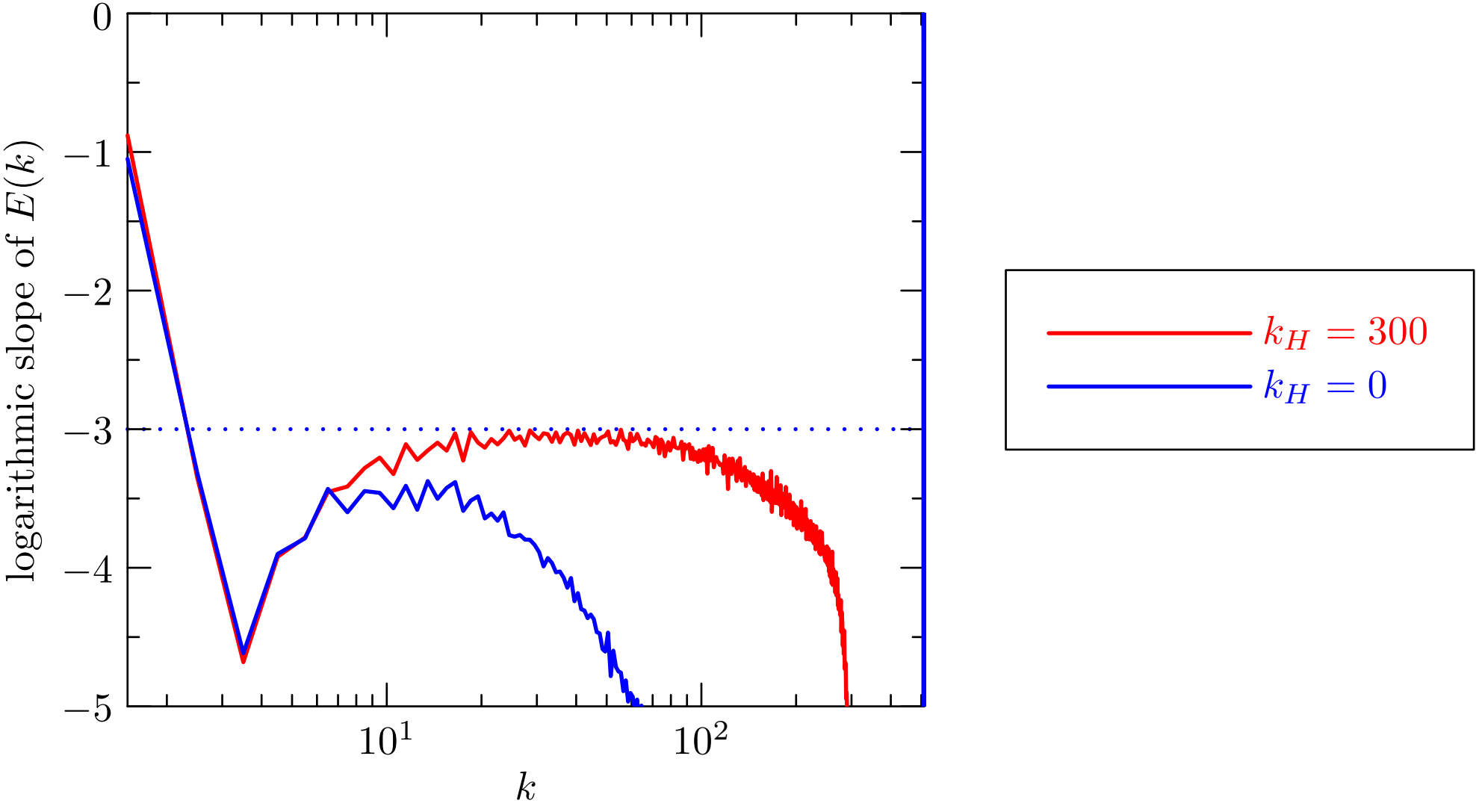
  – sometimes more efficient (by a factor of 2) than pure MPI.

- The use of nonblocking MPI communications allows us to overlap computation with communication: this can yield up to an additional 32% performance gain for implicitly dealiased convolutions, for which a natural parallelism exists between communication and computation.

2D Forced–Dissipative Turbulence Spectrum



2D Forced–Dissipative Power Law Exponent



Conservative Integration

- Conservative integration [SIAM J. Appl. Math 59, 1112 (1999)] provides a useful diagnostic technique for ensuring that the underlying dynamical symmetries have been correctly implemented.

White-Noise Forcing

- The Fourier transform of an isotropic Gaussian white-noise solenoidal force $\boldsymbol{f}$ has the form

$$\boldsymbol{f}_{\boldsymbol{k}}(t) = F_{\boldsymbol{k}}\left(\mathbf{1} - \frac{\boldsymbol{k}\boldsymbol{k}}{k^2}\right)\cdot\boldsymbol{\xi}_{\boldsymbol{k}}(t), \quad \boldsymbol{k}\cdot\boldsymbol{f}_{\boldsymbol{k}} = 0,$$

  where $F_{\boldsymbol{k}}$ is a real number and $\boldsymbol{\xi}_{\boldsymbol{k}}(t)$ is a unit central real Gaussian random 2D vector that satisfies

$$\langle\boldsymbol{\xi}_{\boldsymbol{k}}(t)\boldsymbol{\xi}_{\boldsymbol{k}'}(t')\rangle = \delta_{\boldsymbol{k}\boldsymbol{k}'}\mathbf{1}\delta(t-t').$$

- This implies

$$\langle\boldsymbol{f}_{\boldsymbol{k}}(t)\cdot\boldsymbol{f}_{\boldsymbol{k}'}(t')\rangle = F_{\boldsymbol{k}}^2\delta_{\boldsymbol{k},\boldsymbol{k}'}\,\delta(t-t').$$

White-Noise Forcing

- The rate of energy injection ϵ is given by

$$\epsilon = (\boldsymbol{f}(\boldsymbol{x},t),\boldsymbol{u}(\boldsymbol{x},t)) = \int_\Omega \langle \boldsymbol{f}(\boldsymbol{x},t)\cdot\boldsymbol{u}(\boldsymbol{x},t)\rangle\, d\boldsymbol{x} = \mathrm{Re}\sum_{\boldsymbol{k}} \langle \boldsymbol{f}_{\boldsymbol{k}}(t)\cdot\overline{\boldsymbol{u}}_{\boldsymbol{k}}(t)\rangle$$

- Here $\boldsymbol{u}_{\boldsymbol{k}}(t)$ is functional of the forcing:

$$\boldsymbol{u}_{\boldsymbol{k}}(t) = \boldsymbol{u}_{\boldsymbol{k}'}(t') + \int_{t'}^{t} A_{\boldsymbol{k}}[\boldsymbol{u}(\tau)]d\tau + \int_{t'}^{t} \boldsymbol{f}_{\boldsymbol{k}}(\tau)d\tau,$$

  where $A_{\boldsymbol{k}}$ is a functional of $\boldsymbol{u}$ such that $\frac{\delta A_{\boldsymbol{k}}[\boldsymbol{u}(\tau)]}{\delta \boldsymbol{f}_{\boldsymbol{k}'}(t')}$ is bounded.

- Nonlinear Green's function:

$$\frac{\delta \boldsymbol{u}_{\boldsymbol{k}}(t)}{\delta \boldsymbol{f}_{\boldsymbol{k}'}(t')} = \int_{t'}^{t} \frac{\delta A_{\boldsymbol{k}}[\boldsymbol{u}(\tau)]}{\delta \boldsymbol{f}_{\boldsymbol{k}'}(t')}d\tau + \delta_{\boldsymbol{k}\boldsymbol{k}'}\mathbf{1}H(t-t'),$$

  where H is the Heaviside unit step function.

- To prescribe the forcing amplitude $F_{\boldsymbol{k}}$ in terms of ϵ:

Theorem 1 (Novikov [1964]): *If $f(\boldsymbol{x},t)$ is a Gaussian process, and u is a functional of f, then*

$$\langle f(\boldsymbol{x},t)u(f)\rangle = \int\int \langle f(\boldsymbol{x},t)f(\boldsymbol{x}',t')\rangle \left\langle \frac{\delta u(\boldsymbol{x},t)}{\delta f(\boldsymbol{x}',t')}\right\rangle d\boldsymbol{x}'\,dt'.$$

- For white-noise forcing:

$$\epsilon = \mathrm{Re}\sum_{\boldsymbol{k}} \langle \boldsymbol{f}_{\boldsymbol{k}}(t)\cdot\overline{\boldsymbol{u}}_{\boldsymbol{k}}(t)\rangle = \mathrm{Re}\sum_{\boldsymbol{k},\boldsymbol{k}'}\int \langle \boldsymbol{f}_{\boldsymbol{k}}(t)\overline{\boldsymbol{f}}_{\boldsymbol{k}'}(t')\rangle : \left\langle \frac{\delta\overline{\boldsymbol{u}}_{\boldsymbol{k}}(t)}{\delta\overline{\boldsymbol{f}}_{\boldsymbol{k}'}(t')}\right\rangle dt'$$

$$= \sum_{\boldsymbol{k}} F_{\boldsymbol{k}}^2\left(\mathbf{1}-\frac{\boldsymbol{k}\boldsymbol{k}}{k^2}\right) : \left(\mathbf{1}-\frac{\boldsymbol{k}\boldsymbol{k}}{k^2}\right)H(0)$$

$$= \frac{1}{2}\sum_{\boldsymbol{k}} F_{\boldsymbol{k}}^2,$$

on noting that $H(0) = 1/2$.

Implementation of White-Noise Forcing

- At the end of each time-step, we implement the contribution of white noise forcing with the discretization

$$\omega_{\boldsymbol{k},n+1} = \omega_{\boldsymbol{k},n} + \sqrt{2\tau\eta_{\boldsymbol{k}}}\,\xi,$$

  where ξ is a unit complex Gaussian random number with $\langle\xi\rangle = 0$ and $\left\langle|\xi|^2\right\rangle = 1$.

- This yields the mean enstrophy injection

$$\frac{\left\langle|\omega_{\boldsymbol{k},n+1}|^2 - |\omega_{\boldsymbol{k},n}|^2\right\rangle}{2\tau} = \eta_{\boldsymbol{k}}.$$

Conclusions

- For centered convolutions in d dimensions, implicit padding asymptotically uses $(2/3)^{d-1}$ of the conventional storage.
- The factor of 2 speedup is largely due to increased data locality.
- Highly optimized and parallelized implicit dealiasing routines have been implemented as a software layer `FFTW++` (v 2.05) on top of the `FFTW` library and released under the Lesser GNU Public License: `http://fftwpp.sourceforge.net/`
- The advent of implicit dealiasing of convolutions makes overlapping transposition with FFT computation feasible.
- Dynamic moment averaging allows the integration time window to be specified by the user *a posteriori*.
- Writing a high-performance dealiased pseudospectral code is now a relatively straightforward exercise: skeleton 2D and 3D optimized codes are available at `https://github.com/dealias/dns/tree/master/protodns`.

References

[Basdevant 1983] C. Basdevant, Journal of Computational Physics, **50**:209, 1983.

[Bowman & Roberts 2010] J. C. Bowman & M. Roberts, `FFTW++`: A fast Fourier transform C^{++} header class for the `FFTW3` library, `http://fftwpp.sourceforge.net`, May 6, 2010.

[Bowman & Roberts 2011] J. C. Bowman & M. Roberts, SIAM J. Sci. Comput., **33**:386, 2011.

[Bowman & Roberts 2015] J. C. Bowman & M. Roberts, "Adaptive matrix transpose algorithms for distributed multicore processors," in *Interdisciplinary Topics in Applied Mathematics, Modeling and Computational Science*, edited by M. Cojocaru, I. S. Kotsireas, R. Makarov, R. Melnik, & H. Shodiev, volume 117 of *Springer Proceedings in Mathematics & Statistics*, pp. 97–103, Springer, 2015.

[Novikov 1964] E. A. Novikov, J. Exptl. Theoret. Phys. (U.S.S.R), **47**:1919, 1964.

[Orszag 1971] S. A. Orszag, Journal of the Atmospheric Sciences, **28**:1074, 1971.

[Roberts & Bowman 2018] M. Roberts & J. C. Bowman, J. Comput. Phys., **356**:98, 2018.

[Shadwick *et al.* 1999] B. A. Shadwick, J. C. Bowman, & P. J. Morrison, SIAM J. Appl. Math., **59**:1112, 1999.